Strong Typing

Robert C. Seacord, Software Engineering Institute [vita[1]]

2005-09-27; Updated 2008-10-06

L4 / D/P, L[2]

One way to provide better type checking is to provide better types. Using an unsigned type, for example, can guarantee that a variable does not contain a negative value. However, this solution does not prevent overflow or solve the general case.

Development Context

Integer operations

Technology Context

C, C++, IA-32, Win32, UNIX

Attacks

Attacker executes arbitrary code on machine with permissions of compromised process or changes the behavior of the program.

Risk

Integers in C and C++ are susceptible to overflow, sign, and truncation errors that can lead to exploitable vulnerabilities.

Description

One way to provide better type checking is to provide better types. Using an unsigned type, for example, can guarantee that a variable does not contain a negative value. However, this solution does not prevent overflow or solve the general case.

Data abstractions can support data ranges in a way that standard and extended integer types cannot. Data abstractions are possible in both C and C++, although C++ provides more support. For example, if an integer was required to store the temperature of water in liquid form using the Fahrenheit scale, we could declare a variable as follows:

unsigned char waterTemperature;

Using waterTemperature to represent an unsigned 8-bit value from 1–255 is sufficient; water ranges from 32 degrees Fahrenheit (freezing) to 212 degrees Fahrenheit (the boiling point). However, this type does not prevent overflow and also allows for invalid values (that is, 1–31 and 213–255).

One solution is to create an abstract type in which waterTemperature is private and cannot be directly accessed by the user. A user of this data abstraction can only access, update, or operate on this value through public method calls. These methods must provide *type safety* by ensuring that the value of the waterTemperature does not leave the valid range. If this is done properly, there is no possibility of an integer type range error occurring.

This data abstraction is easy to write in C++ and C. A C programmer could specify `create()` and `destroy()` methods instead of constructors and destructors but would not be able to redefine operators. Inheritance and other features of C++ are not required to create usable data abstractions.

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/274-BSI.html (Seacord, Robert C.)

Pearson Education, Inc. Copyright